P2 Design-Test Document

Partner A: Aaron Kulkarni

Partner B: Dev Randalpura

1. Server Design

Overview

Describe the overall server design. You must include the number of threads you create including the main thread, what each thread is for and doing, any loops you have for each thread (and do this for both event-loop and thread-based server).

We did not have any threads. The server was created with an event loop.

Justification

Justify why your design is efficient and show evidence it can handle load efficiently. Specify the number of clients, the file size for each client, and the response rate/delay each client.

The design of the server is efficient because sessions and the server are ended instantly after an error, with no added waiting time.

The server design can handle any number of clients. There is not a limit to the file size for each client. There is a reasonable delay.

Data structures

List any notable data structures you used and justify the use of the data structure. Also specify how you handle synchronization for the data structure if there were any need for synchronization.

There were a few data structures that we used. There was the event loop, along with the socket and keyboard data structures that would allow for that functionality. Other than that, we had two data structures to deal with sessions. First, we had a class called “Session”. When an object was instantiated with this class, it would hold all the needed information for a session. Also, we had a dictionary of sessions, which mapped Session Id’s to the appropriate session object.

How timeouts are handled

Describe how the timeouts are handled.

When a timeout happens, the function “timeoutSession” is called. timeoutSession goes through the dictionary of session ids, finds the correct session, then deletes the object and removes it from the dictionary. It also sends a goodbye message to the client for that session.

How shutdown is handled

Describe how you handled the shutdown gracefully. (That is when you hit 'q' or ctrl+d in the command prompt.)

When the server receives a “q”, it calls the function endServer. endServer loop through the session dictionary, ending each currently active session and sending goodbye to the clients. Then, it ends the loop and exits the program.

Any libraries used

List any libraries that you used for the implementation and justify the usage.

No libraries were used.

Corner cases identified

Describe corner cases that you identified and how you handled them.

There were no particularly unique corner cases.

2. Server Testing

Testing your server

Describe how you tested your server. Include the description of each test case and the setting (such as local only, one local one remote, how many clients, use of mock client, etc.).

Basic testing was done by manually creating a single client and sending information to the server. Then, we would test out manual timeouts and session endings. After basic testing was done, we would try one local client and one remote client.

3. Client Design

Overview

Describe the overall client design. Specify any differences between event-loop based client design and thread-based client design. You must include the number of threads you create including the main thread, what each thread is for and doing, any loops you have for each thread (and do this for both event-loop and thread-based client).

We created three threads(other than the main thread). One thread was for handling keyboard input. One thread was for handling socket communication. The final thread was the timer thread which was created and destroyed multiple times throughout the program. The keyboard and socket thread shared a boolean variable called “keepGoing”. When one of them wanted to end, they would set keepGoing to false to signal to the other thread that it should stop working.

Justification

Justify why your design above is efficient. Also show evidence how your client can handle sending packets from a large file (each line is close to UDP packet max and also contains many lines) and receiving packets from the server at the same time. Note you do not want to cause the false TIMEOUT from server because you are too busy just sending out the packets to the server when the server actually has sent you a packet before the TIMEOUT.

Since there are two threads handling everything independently, there was no problem sending and receiving packets at the same time. After sending out a packet, the socket threads waits for a response from the server before sending out another packet.

Data structures

List any notable data structures you used and justify the use of the data structure. Also specify how you handle synchronization for the data structure if there were any need for synchronization.

To handle communication between keyboard and socket, we created a queue. This queue was a queue of lines that the keyboard had received and wanted to send out to the server. Whenever the queue is non-empty, the server pops the top element from the queue and sends it to the server.

How timeouts are handled

Describe how the timeouts are handled.

When a timeout occurs, the function “closing” is called, which sends the client into the closing state. In the closing state, the client sends a goodbye message to the server and waits for a goodbye message from the server.

How shutdown is handled

Describe how you handled the shutdown gracefully. (That is when you hit 'q' or ctrl+d in the command prompt.)

When a shutdown happens, the socket is closed and the program exits.

Any libraries used

List any libraries that you used and justify the usage.

We used the queue library, for the queue between socket and keyboard.

We used the random library, for assigning a random session id to the client.

We used the time library to deal with an edge case identified in the following section.

Corner cases identified

Describe corner cases that you identified and how you handled them.

One edge case that we came across dealt with receiving a goodbye from the server. Generally, our program sends out a packet and only then waits for a response from the server. But if there is nothing in the queue and no messages to send out, and the server sends a goodbye message, we would have never been looking to receive it. So to fix this, we wrote some code to continuously receive from the socket in a non-blocking manner when there is nothing in the queue. We used the time library to pause the thread for a short period of time after each empty receive call.

4. Client Testing

Testing your client

Describe how you tested your client. Include the description of each test case and the setting (such as local only, one local one remote, how many clients, use of mock server, etc.).

We did basic testing on the client, where we would send messages to the server and see if they showed up there. Also, we would attempt to force a server timeout in order to see the clients response.

5. Reflection

Reflection on the process

What was most challenging in implementing the server? What was most challenging testing the server?

Most challenging part of the server implementation was dealing with the timeouts and finding a way to integrate the timer functionality in pyuv into our code. There were no real challenges in testing the server.

What was most challenging in implementing the client? What was most challenging testing the client?

The most challenging part of implementing the client was dealing with edge cases regarding sending out messages and our queue. Also, the complex diagram with the "closing" and "closed" state confused us at the beginning. Most challenging part of testing the client was going through all the possible edge cases.

What was most fun working on the project? What was most not-so-fun?

Most fun part of working on the project was writing the base code. The not-so-fun part was dealing with the edge cases and timer/timeouts.

If you are to do this all over again how would you do it differently?

If we were to do this all over again, we would take a more incremental approach.

Reflection on pair programming

Log of the amount of time spent driving and the amount of time spent working individually for each part (e.g., X drives 1 hour; Y drives 45 minutes; X works alone for 1 hour, etc.)

We did not keep a log of the amount of time spent working. Generally, we would work for a few hours at a time together, alternating drivers. Then, after each work session, we would spend about 15-30 minutes making minor changes on our own.

What went well/or not-so-well doing pair programming? What was your take away in this process?

Pair programming allowed for a lot of idea bouncing, which made the process more enjoyable and quicker overall. Our takeaway was to consult the other person

whenever we faced a roadblock, as that generally allowed us to resolve it and keep going.

Submission

Remember to export to pdf and push it to your github team repo under the project root (the same level as README.txt)